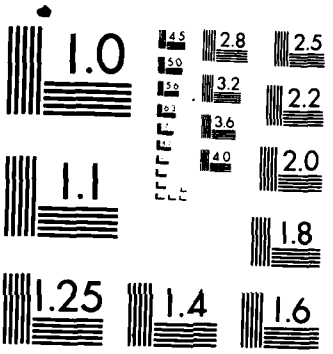
AD-A161 391 FAST AND EFFICIENT ALGORITHMS FOR LINEAR PROGRAMMING AND FOR THE LINEAR L (U) HARVARD UNIV CAMBRIDGE MA AIKEN COMPUTATION LAB V PAN ET AL 1985 TR-11-85 1/1

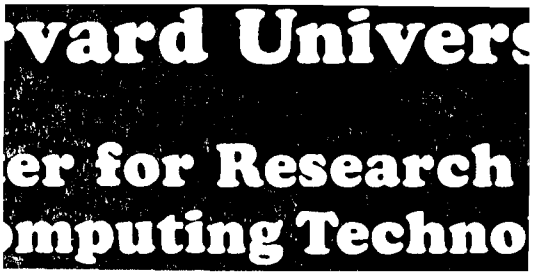
UNCLASSIFIED N00014-80-C-0647 F/G 12/1 NL

END

FILMED

DTIC




vard Univers
er for Research
mputing Techno

12

# FAST AND EFFICIENT ALGORITHMS FOR LINEAR PROGRAMMING AND FOR THE LINEAR LEAST SQUARES PROBLEM

Victor Pan
and
John Reif

TR-11-85

unclassified

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

| REPORT DOCUMENTATION PAGE | | READ INSTRUCTIONS BEFORE COMPLETING FORM |
|---|---|---|
| 1. REPORT NUMBER | 2. GOVT ACCESSION NO. AD-A160 139 | 3. RECIPIENT'S CATALOG NUMBER |
| 4. TITLE *(and Subtitle)* FAST AND EFFICIENT ALGORITHMS FOR LINEAR PROGRAMMING AND FOR THE LINEAR LEAST SQUARES PROBLEM | | 5. TYPE OF REPORT & PERIOD COVERED Technical Report |
| | | 6. PERFORMING ORG. REPORT NUMBER TR-11-85 |
| 7. AUTHOR(s) Victor Pan, John H. Reif | | 8. CONTRACT OR GRANT NUMBER(s) N00014-80-C-0647 |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS Harvard University, Cambridge, MA 02138 | | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS |
| 11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research, 800 North Quincy Street, Arlington, VA 22217 | | 12. REPORT DATE 1985 |
| | | 13. NUMBER OF PAGES 14 |
| 14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)* Same as above | | 15. SECURITY CLASS. *(of this report)* |
| | | 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE |

16. DISTRIBUTION STATEMENT *(of this Report)*

unlimited

This document has been approved for public release and sale; its distribution is unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

unlimited

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

nested dissection, linear programming, Karmarkar's algorithm, grid graphs, planar graphs, least squares, sparse linear system, parallel algorithms

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

See reverse side

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

We present a new parallel algorithm for computing a least-squares solution to a sparse overdetermined system of linear equations $Ax=b$ such that $m \times n$ matrix A is sparse and the graph, $G = (V,E)$, of the matrix $H = \begin{bmatrix} I & A^T \\ A & 0 \end{bmatrix}$ has an $s(m+n)$-separator family, that is, by deleting a separator subset S of vertices of the size $\leq s(m+n)$, G can be partitioned into two disconnected subgraphs having vertex sets $V_1, V_2$ of the sizes $\leq 2/3\,(m+n)$ and each of the two resulting subgraphs induced by the vertex sets $S \cup V_i$, $i=1,2$, can be recursively $s(|S \cup V_i|)$-separated in a similar way. Our algorithm uses $O(\log (m+n) \log^2 s(m+n))$ steps and $\leq s^3(m+n)$ processors; it relies on our recent parallel algorithm for solving sparse linear systems and has several immediate

Appeared in 12th International
Symposium on Mathematical
Programming, MIT, Cambridge, MA
August 1985

# Fast and Efficient Algorithms for Linear Programming and for the Linear Least Squares Problem

*Victor Pan* *

Computer Science Department
State University of New York at Albany
Albany, New York
and

*John Reif* **

Aiken Computation Lab.
Division of Applied Sciences, Harvard University
Cambridge, MA



## Summary ***

We present a new parallel algorithm for computing a least-squares solution to a sparse overdetermined system of linear equations $\mathbf{Ax=b}$ such that $m \times n$ matrix $A$ *is sparse and the graph,* $G = (V,E)$, *of the matrix* $H = \begin{bmatrix} I & A^T \\ A & 0 \end{bmatrix}$ has an $s(m+n)$-separator family, that is, by deleting a separator subset $S$ of vertices of the size $\leq s(m+n)$, $G$ can be partitioned into two disconnected subgraphs having vertex sets $V_1, V_2$ of the sizes $\leq 2/3\,(m+n)$ and each of the two resulting subgraphs induced by the vertex sets $S \cup V_i$, $i=1,2$, can be recursively $s(|S \cup V_i|)$-separated in a similar way. Our algorithm uses $O(\log (m+n) \log^2 s(m+n))$ steps and $\leq s^3(m+n)$ processors; it relies on our recent parallel algorithm for solving sparse linear systems and has several immediate



* Supported by NSF Grants MCS 8203232 and DCR 8507573.

** This work was supported by Office of Naval Research Contract N00014-80-C-0647.

*** This paper was presented at the 12-th International Symposium on Mathematical Programming, August 5-9, 1985, Boston, Massachusetts.

applications of substantial interest, in particular to mathematical programming, to sparse nonsymmetric systems of linear equations, and to the path algebra computation. We most closely examine the impact on *the linear programming problem,* which requires to maximize $\mathbf{c}^T\mathbf{y}$ subject to $\mathbf{A}^T\mathbf{y} \leq \mathbf{b}$, $\mathbf{y} \geq \mathbf{0}$ where A is an $m \times n$ matrix. Hereafter it is assumed that $m \geq n$. The recent algorithm by N. Karmarkar gives the best known upper estimate, ($O(m^{3.5} L)$ arithmetic operations where L is the input size), for the cost of the solution of this problem in the worst case. We prove an asymptotic improvement of that result in the case where the graph of the associated matrix H has an $s(m+n)$-separator family; then our algorithm can be implemented using $O(m\, L \log m \log^2 s(m+n))$ parallel arithmetic steps, $s^3(m+n)$ processors and a total of $O(m\, L\, s^3(m+n) \log m \log^2 s(m+n))$ arithmetic operations. In many cases of practical importance this is a considerable improvement of the known estimates: for example, $s(m+n) = \sqrt{8\,m+n}$ if G is planar, (as occurs in many operations research applications, for instance, in the problem of computing the maximum multicommodity flow with a bounded number of commodities in a network having an $s(m+n)$-separator family), so that the processor bound is only $8\sqrt{8}\,(m+n)^{1.5}$ and the total number of arithmetic steps is $O((m+n)^{2.5}L)$ in that case. Similarly Karmarkar's algorithm and the known algorithms for the solution of overdetermined linear systems are accelerated in the case of dense input matrices via our recent parallel algorithms for the inversion of dense $n \times n$ matrices using $O(\log^2 n)$ steps, $n^3$ processors. Combined with Karmarkar's algorithm, this implies $O(L(m+n) \log^2(m+n))$ steps, $n^3$ processors. The stated results promise some important practical applications. Theoretically the above processor bounds can be reduced to $o(n^{2.5})$ and $o((m+n)^{2.5})$ in the dense case and to $o(s^{2.5}(m+n))$ in the sparse case (supporting the same number of parallel steps).

## 1. Introduction

Numerous practical computations require to find a least squares solution to an *overdetermined* system of linear equations, $Ax = b$, that is, to find a vector $x$ of dimension $n$ that minimizes $||Ax - b||$ given an $m \times n$ matrix $A$ and a vector $b$ of dimension $m$ where $m \geq n$. (Here and hereafter we apply the Euclidean vector norm and the associated 2-norm of matrices, [GL].) Such a problem is called the *linear least squares problem, l.l.s.p.* In particular solving a linear system $Ax = b$ in the usual sense is a simplification of the l.l.s.p. where the output is either the answer that $\min_x ||Ax - b|| > 0$ or otherwise a vector $x^*$ such that $Ax^* - b = 0$.

The objective of this paper is to reexamine the time-complexity of the l.l.s.p. and to indicate the possibility of speeding up its solution using the parallel algorithms of [PR]. As a major consequence, (which may become decisive for determining the best algorithm for the *linear programming problem (l.p.p.)*, at least over some important classes of instances of that problem, see Appendix), *we will substantially speed up Karmarkar's algorithm*, [K], for the l.p.p. because solving the l.l.s.p. constitutes the most costly part of every iteration of that algorithm. Our acceleration of Karmarkar's is most significant in the practically important case, (arising, for instance, in the multicommodity flow problem in a planar network for a fixed number of commodities, see [GM], [L]), where the input matrix of l.p.p. is large and sparse and is associated with graphs having small separators (see the formal definition below, in sect. 3). On the other hand, our work has several further impacts. Similarly to the case of the algorithm of [K], we may immediately improve the performance of several known algorithms, in particular of the algorithms for systems of linear inequalities, [P84,85], for mathematical programming, [S], and for sparse nonsymmetric systems of linear equations for, (as we indicated above), solving a system of linear equations constitutes a

particular case of the l.l.s.p. where $\min_{\mathbf{x}} ||A\mathbf{x} - \mathbf{b}|| = 0$). The latter observation leads to a very wide range of applications of our results, including in particular the *acceleration of the simplex algorithms,* see [C], [M], for sparse l.p. problems. Further applications may include several combinatorial computations. This is demonstrated in [PRa] where, relying on the latter improvement of the algorithms for sparse nonsymmetric systems of linear equations, we extend the parallel nested dissection algorithm of [PR] to the path algebra computations.

We organize the paper as follows. In the next section we recall two known representations of the l.l.s.p. using normal equations. In sect. 3 we reexamine the computational cost of sequential algorithms for l.l.s.p., in particular, we recall the sequential nested dissection algorithm of [LRT] and adjust it to the case of l.l.s.p. In sect. 4 we estimate the cost of performing our parallel algorithm for the same problem. In sect. 5 we consider one of the major applications of our results, that is, to the acceleration of Karmarkar's algorithm. In Appendix we will briefly comment on the current estimates for the computational cost of solving the l.p.p.

## 2. Some Equivalent Representations of the Linear Least Squares Problem

We will use the known fact, (see [GL]), that the l.l.s.p. can be reduced to computing solution $\mathbf{x}$ to the system of normal linear equations

$$A^T A\,\mathbf{x} = A^T\mathbf{b}, \tag{1}$$

which can be equivalently rewritten as the following system of linear equations in $\mathbf{r}$ and $\mathbf{x}$,

$$H(\mathbf{r},\mathbf{x})^T = (\mathbf{b},0)^T \tag{2}$$

where

$$H = \begin{pmatrix} \beta I & A \\ A^T & O \end{pmatrix}, \tag{3}$$

$\beta$ is a nonzero constant, see [B], p. 182. Here and hereafter I, $W^T$,$\mathbf{v}^T$, O and $\mathbf{0}$ denote the identity matrix, the transposes of a matrix W and of a vector $\mathbf{v}$, the null matrix and the null vector, respectively.

If we need to solve the system $A\mathbf{x} = \mathbf{b}$ in the usual sense, then that system can be equivalently rewritten as $CA\mathbf{x} = C\mathbf{b}$ for any nonsingular matrix C, so we may generalize (2), (3) as follows,

$$H(C)(\mathbf{r},\mathbf{x})^T = (C\mathbf{b},\mathbf{0})^T,\ H(C) = \begin{bmatrix} \lambda I & CA \\ A^TC & O \end{bmatrix} . \tag{4}$$

Here $\lambda$, $\mu$ are constants, not necessarily nonzero. For given A and $\mathbf{b}$, we may vary $\beta$, $\lambda$, $\mu$ and C in (3) and (4) in order to simplify the problem or to improve the stability of its solution (obtained, say by the nested dissection algorithms considered in the next sections).

## 3. The Sequential Computational Complexity of the Linear Least Squares Problem

For a l.l.s.p. with a dense matrix A, its solution can be obtained from (1) using O(m/n)M(n) arithmetic operations where M(n) is the cost of $n \times n$ matrix multiplication, $M(n) \leq 2n^3-n^2$. Theoretically $M(n) = o(n^{2.496})$ but that bound is not practical due to the huge overhead constants hidden in that "o", [P84a].

If the matrix A is sparse, the solution can be accelerated using some special methods, see [B]. In particular applying the conjugate gradient method or the Lanczos method, (see [B], [GL]), we may reduce the cost of solving both the system (1) and (consequently) a l.l.s.p. at least to O(n N(A)) arithmetic operations where N(A) is the number of nonzero entries of A, provided that the multiplication by 0 and the addition of 0 are cost-free operations.

We will single out a more specific case encountered in many practical instances of the l.l.s.p., that is, in the instances where the matrix A is sparse and

furthermore the graph G=(V,E) associated with the matrix H has an s(m+n)-separator family where s(m+n)=o(m+n). Here and hereafter we follow the recursive Definition 1.1 of sect. 1.2 of [PR], (compare also [LRT]), saying that G has an s(m+n)-separator family if G can be partioned into two disconnected subgraphs with the vertex sets $V_1$ and $V_2$ of the sizes $\leq \alpha \mid V \mid$, (for a constant $\alpha < 1$), by deleting a separator subset S of vertices of the size $\leq$ s(m+n) and if recursively each of the two subgraphs of G induced by the vertex sets $S \cup V_i$, i=1,2, has an s( $\mid S \cup V_i \mid$ )-separator family.

In that case the application of the techniques of nested dissection, (see [G], [LRT], [B], p. 182), decreases the cost of the solution of (2), and consequently of the original l.l.s.p., to O( $\mid E \mid$ +M(s(m+n))) arithmetic operations where |E| is the cardinality of the edge set of G, [LRT]. This is the cost of computing the $LDL^T$-factorization of H. The subsequent evaluation of the vectors **r**,**x** satisfying (2) costs only O( $\mid E \mid +(s(m+n))^2$) arithmetic operations so the approach is particularly effective where several systems (1) with fixed A and variable **b** must be solved.

## 4. Acceleration of the Solution of Linear Least Squares Problem Using Parallel Algorithms.

For large input matrices A the sequential algorithms for the l.l.s.p. can be prohibitively slow. Their drastic acceleration that fully preserves their efficiency can be obtained using the recent parallel algorithms of [PR] where in each step every processor may perform one arithmetic operation. (Note that we need fewer processors where we agree to use more steps.) Specifically, applying the matrix inversion algorithm of [PR], we solve the system (1) in $O(\log^2 n)$ parallel steps using M(n)/log n processors and consequently we solve the original l.l.s.p. using $O(\log m + \log^2 n)$ steps, (M(n)/log n)(1+m/(n log n)) processors. These are the bounds in the case where A is a general (dense) matrix. If A is sparse and if the

graph G = (V,E) of H has an s(m+n)-separator family, then the parallel nested dissection algorithm of [PR] computes a special recursive s(m+n)-factorization of the matrix H of (2),(3) using $O(\log m \log^2 s(m+n))$ parallel steps and $|E| + M(s(m+n))/\log s(m+n)$ processors. Following Definition 4.1 of [PR], we define such a recursive s(m+n)-factorization of H as a sequence of matrices $H_0, H_1, \ldots, H_d$ such that $H_0 = PHP^T$, P is an $n \times n$ permutation matrix,

$$H_g = \begin{bmatrix} X_g & Y_g^T \\ Y_g & Z_g \end{bmatrix}, \quad Z_g = H_{g+1} + Y_g X_g^{-1} Y_g^T \tag{6}$$

for g=0,1,...,d-1, and $X_g$ is a block-diagonal matrix consisting of square blocks of sizes at most $s(\alpha^{d-h}(m+n)) \times s(\alpha^{d-h}(m+n))$ where $\alpha^d(m+n) \leq c$ for a constant c. The factorization (6) has length $d = O(\log m)$, so its computation is reduced to $O(\log m)$ steps of matrix multiplication and inversion versus m+n such steps of the sequential nested dissection algorithms. Although such a recursive factorization (6) is distinct from the more customary factorization used in the sequential algorithms, *both have similar power*, that is, when the recursive factorization (6) is available, $O((\log m)(\log s(m+n)))$ parallel steps and $|E| + (s(m+n))^2$ processors suffice in order to solve the system (2) and consequently the original l.l.s.p.

Comparing the cost bounds of [LRT] and [PR] we can see that the parallelization is fully efficient, that is, the product of the two upper bounds on the numbers of steps and processors of [PR] is equal (within a polylogarithmic factor) to the bound on the number of arithmetic operations in the current best sequential algorithm of [LRT] for the same problem. The same efficiency criterion is satisfied in the algorithms of [PR] inverting an $n \times n$ dense matrix in $O(\log^2 n)$ parallel steps using $M(n)/\log n$ processors. Consequently all our parallel algorithms for a l.l.s.p. are also fully efficient.

The complexity estimates of [PR] have been established in the case of well-conditioned input matrices; the algorithms of [PR] output the approximate solu-

tions with a sufficiently high precision. On the other hand, all the estimates have been extended to the case of arbitrary integer input matrices in [P85a] using some different techniques. In that case the solutions are computed exactly.

## 5. Parallelization of Karmarkar's Algorithm and Its Application to Sparse Linear Programming.

In this section we will examine the cost of Karmarkar's linear programming algorithm, [K], and of its improvements that use the parallelization and the nested dissection. At first we will reproduce that algorithm, which solves the problem of the minimization of the linear function $\mathbf{c}^T\mathbf{y}$ subject to the constraints

$$A^T\mathbf{y} = \mathbf{0}, \sum_j y_j = 1, \mathbf{y} \geq \mathbf{0}, \tag{7}$$

where $\mathbf{y} = [y_j, j=0,1,...,m-1]$ and $\mathbf{c}$ are m-dimensional vectors, $A^T$ is an $n \times m$ matrix, $m \geq n$, $\mathbf{y}$ is unknown. This version is equivalent to the canonical linear programming problem of the minimization of $\mathbf{c}^T\mathbf{y}$ subject to $A^T\mathbf{y} \leq \mathbf{b}$, $\mathbf{y} \geq \mathbf{0}$, see [K] and compare [C], [M]. We will designate $\mathbf{e} = [1,1,....,1]^T$,

$$\begin{gathered} \mathbf{y}(i) = [y_0(i), y_1(i), ...., y_{m-1}(i)], \\ D(0) = I,\ D(i) = \mathrm{diag}\,(y_0(i), y_1(i), ...., y_{m-1}(i)), \\ B^T = B^T(i) = \begin{bmatrix} A^T D(i) \\ \mathbf{e}^T \end{bmatrix},\ i=0,1,... \end{gathered} \tag{8}$$

(All the matrices D(i) encountered in the algorithm of [K] are nonsingular.) The algorithm proceeds as follows.

**Initialize.** Choose $\epsilon > 0$ (prescribe tolerance) and a parameter $\beta$ (in particular, $\beta$ can be set equal to 1/4). Let $\mathbf{y}(0) = (1/n)\mathbf{e}, i=0$.

**Recursive Step.** While $\mathbf{c}^T\mathbf{y}(i) > \epsilon$ do

Compute the vector $\mathbf{y}(i+1) = \gamma(\mathbf{y}(i))$, increment i.

Given vector $\mathbf{y}(i)$, the vector $\mathbf{y}(i+1)$ is computed as follows.

1. Compute the matrix $B = B(i)$ of (8), that is, compute the matrix $A^T D(i)$ and

augment it by appending the row $\mathbf{e}^T$.

2. Compute the vector $\mathbf{c}_p = [I - B(B^TB)^{-1}B^T]D(i)\mathbf{c}$.

3. Compute the vector $\mathbf{z}(i) = \mathbf{y}(0) - \beta r\mathbf{c}_p / ||\mathbf{c}_p||$ where $r = 1/\sqrt{m(m-1)}$.

4. Compute the vector $\mathbf{y}(i+1) = D(i)\mathbf{z}(i) / \mathbf{e}^TD(i)\mathbf{z}(i))$.

The algorithm also includes the checks for infeasibiity and optimality, (see [K]), but it is easy to verify that their computational costs, as well as the computational cost of the reduction of the problem from the canonical form to (7), are dominated by the cost of computing the vector $\gamma(\mathbf{y}(i))$ at the recursive steps, which is, in turn, dominated by the cost of computing $B^TB$ given $B^T = B^T(i)$ for all i. [K] shows that $B^TB$ can be represented as follows,

$$B^TB = \begin{bmatrix} A^TD^2(i)A & \mathbf{0} \\ \mathbf{0}^T & m \end{bmatrix},$$

so the inversion of $B^TB$ is reduced to the inversion of $A^TD^2(i)A$, which in turn is reduced to the inversion of the matrix H of (2),(3) where A is replaced by D(i)A. Furthermore we can see that it suffices to compute the product $(B^TB)^{-1}BD(i)\mathbf{c}$, and this amounts to matrix-vector multiplications and to solving a system of linear equations with the matrix H.

This algorithm of [K] requires O(Lm) recursive steps in the worst case, so the total computational cost is O(Lm C) where L is the input size of the problem and C is the cost of computing $\gamma(\mathbf{y})$ given $\mathbf{y}$. The algorithm for the incremental computation of the inverse of $B^TB$ of sect. 6 of [K] implies that $C = O(m^{2.5})$ for the dense A. It is rather straightforward to perform these $O(m^{2.5})$ arithmetic operations in parallel using $O(\sqrt{m}\log m)$ steps and $m^2/\log m$ processors (and using O(m) steps, $m^2$ processors for the initial inversion of $A^TA$). Applying the matrix inversion algorithms of [PR], we *may perform* every evaluation of $\gamma(y)$ using $O(\log m + \log^2 n)$ parallel arithmetic steps and $(M(n)/\log n)(1+m/(n\log n))$ processors, so we arrive at the following trade-off for the estimated total arithmetic

cost of Karmarkar's algorithm, $O(m^{1.5}L)$ steps, $m^2$ processors, that is, $O(m^{3.5}L)$ arithmetic operations, (via the straightforward parallelization), or $O(m\,L(\log m+\log^2 n))$ steps, $(M(n)/\log n)(1+m/(n \log n))$ processors, that is, $O(m\,L\,M(n)(\log m+\log^2 n)(1+m/(n \log n))/\log n)$ arithmetic operations, (via the parallel matrix inversion algorithms of [PR]).

In both cases the sparsity of A is not exploited. In particular the algorithm for the incremental computation of the inverse suggested in sect. 6 of [K] does not preserve the sparsity of the original input matrix. This causes some difficulties for the practical computation for the storage space increases substantially. Thus the special methods of solving sparse l.l.s.p., such as the conjugate gradient, the Lanczos and the nested dissection methods, (see [B], [GL] and this paper), become competitive with (if not superior to) the latter algorithm of sect. 6 of [K]. If the matrix A is such that the graph $G=(V,E)$ of the matrix H of (3) has an $s(m+n)$-separator family and $s(m+n)=o(m+n)$, then the nested dissection method can be strongly recommended. Specifically in this case we arrive at the estimates of $O(Lm(|E| + M(s(m+n)))$ arithmetic operations by combining [K] and [LRT], see our sect. 3, and of $O(Lm \log m \log^2 s(m+n))$ parallel arithmetic steps and $O(|E| +M(s(m+n))/\log s(m+n))$ processors by combining [K] and the parallel algorithm of this paper. The reader could better appreciate this improvement due to the application of nested dissection if we recall that $s(m+n) = \sqrt{8(m+n)}$ if the graph G is planar (as occurs in many operations research applications, for instance, in the problem of computing the maximum flow in a network having an $s(m+n)$-separator family). Then the processor bound for computing the recursive factorization (6) is less than $2s^3(m + n) = 8\sqrt{8}\,(m+n)^{1.5}$ and the total number of arithmetic operations is $O(m\,L(m+n)^{1.5})$ in that case. The premultiplications of A by the nonsingular matrix D(i) do not change the separator sets for the graph G, so these sets are

precomputed once and for all, which is an additional advantage of using the nested dissection in this case.

## Appendix. Current Computational Cost of Solving the Linear Programming Problem.

We will start with the table presenting estimates for the computational cost of one iteration of the simplex and Karmarkar's algorithms for the l.p.p. having a dense $m \times n$ input matrix A, compare [P85b], [PR] and sect. 5. We will restrict our analysis to the cases where $n \leq m = O(n)$.

Table 1.

| | arithmetic operations | parallel steps | processors |
|---|---|---|---|
| 1-st iteration of [K] | $O(m^3)$ | $O(\log^2 m)$ | $m^3/\log m$ |
| average over n iterations of [K] | $O(m^{2.5})$ | $O(\sqrt{m} \log m)$ | $m^2$ |
| any iteration of revised simplex algorithms | $O(m^2)$ | $O(m)$ | $m$ |

There is a certain controversy about the current upper estimates for the number of iterations in the two cited algorithms. The worst case upper bounds, $O(Lm)$ for [K] and $2^m$ for the simplex algorithms, greatly exceed the number of iterations required where the same algorithms run in practice or use random input instances, (similarly for the algorithms of [P84,85], see [MP]). This uncertainty complicates the theoretical comparison of the effectiveness of the two algorithms. However, some preliminary comparison can be based on the partial

information already available. In particular let us assume the empirical upper bound O(n log m) on the number of iterations (pivot steps) of the simplex algorithms, cited by some authors who refer to the decades of practical computation, see [C], pp. 45-46, [M], p. 434. The bound implies that a total of $O(m^3 \log m)$ arithmetic operations suffice in the simplex algorithm vs. $O(m^3)$ used already in the first iteration of [K]. Moreover there are special methods that efficiently update the triangular factorization of the basis matrices used in the simplex algorithms, which further simplifies every iteration of the simplex algorithms in the case of sparse input matrices, see [C], ch. 7,24, [M], ch. 7. On the other hand, if *appropriate modifications* of Karmarkar's *original* algorithm indeed run in a polylogarithmic number of iterations (as he reported on the TIMS/ORSA meeting, Boston, May, 1985, and on the 12-th International Symposium on Mathematical Programming, Boston, August, 1985), this would immediately imply a substantial acceleration of the simplex algorithm at least in the cases of i) parallel computation and dense input matrices (see Table 1), and ii) both parallel and sequential computations in the cases where the graph associated with the matrix H of (2) has an s(m+n)-separator family, $s(m+n) = O((m+n)^q)$, $q < 1$, (see the estimates of our sect. 5).

## References


[B] Å. Björck 1976, Methods for Sparse Linear Least Squares Problems, 177-200, in *Sparse Matrix Computations* (J.R. Bunch and D.J. Rose edits.), Academic Press, N.Y.

[C] V. Chvatal 1983, *Linear Programming*, W.H. Freeman, New York-San Francisco.

[G] J.A. George 1973, Nested Dissection of a Regular Finite Element Mesh, *SIAM J. on Numerical Analysis*, **10**,2, 345-367.

[GL] G.H. Golub and C.F. van Loan 1983, *Matrix Computations*, the Johns Hopkins Univ. Press, Baltimore, Maryland.

[GM] M. Gondran and M. Minoux 1984, *Graphs and Algorithms*, Wiley-Interscience, New York.

[K] N.K. Karmarkar 1984, A New Polynomial Time Algorithm for Linear Programming, *Combinatorica* **4**,4, 373-395.

[L] E. Lawler 1976, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, N.Y.

[LRT] R. Lipton, D. Rose and R.E. Tarjan 1979, Generalized Nested Dissection, *SIAM J. on Numerical Analysis* **16**,2, 346-358.

[MP] R. Mewing and V. Pan 1985, Practical Improvements of Recent Finite Algorithms for a System of Linear Inequalities, manuscript.

[M] K.G. Murty 1983, *Linear Programming*, Wiley, New York.

[P84] V. Pan 1984, How Fast Can We Solve a System of Linear Inequalities in the Worst and in the Average Cases?, *Methods of Operations Research* **51**, 107-118.

[P84a] V. Pan 1984, *How to Multiply Matrices Fast*, Lecture Notes in Computer Science **179**, Springer-Verlag, Berlin.

[P85] V. Pan 1985, Fast Finite Methods for a System of Linear Inequalities, *Computers and Mathematics (with Applics.)* **11**,4, 355-394.

[P85a] V. Pan 1985, Fast and Efficient Algorithms for the Exact Inversion of Integer Matrices, Tech. Report 85-2, *Computer Sci. Dept., SUNYA* (April 1985).

[P85b] V. Pan 1985, On the Complexity of a Pivot Step of Revised Simplex Algorithm, *Computers and Mathematics (with Applics.)*, in print.

[PR] V. Pan and J. Reif 1984, Fast and Efficient Solution of Linear Systems, Tech. Report TR-02-85, *Center for Research in Computer Technology, Aiken Computation Laboratory, Harvard Univ.* Cambridge, Mass., (short version in *Proc. 17-th Ann. ACM STOC*, 143-152, Providence, R.I.).

[PRa] V. Pan and J. Reif 1985, Extension of the Parallel Nested Dissection Algorithm to the Path Algebra Problems, Tech. Report 85-9, *Computer Science Dept. SUNY Albany* (June 1985).

[S] N.Z. Shor 1977, New Development Trend in Nondifferentiable Optimization, *Kibernetika* **13**,6, 87-91, (transl. in *Cybernetics* **13**,6, 881-886 (1977)).

# END

# FILMED

1-86

# DTIC